

# Quick Introduction to IPython Notebook

## Modules to load on discover

```
Modules to load on discover:
```

- 1) other/comp/gcc-4.5-sp1
- 2) lib/mkl-10.1.2.024
- 3) other/SIVO-PyD/spd\_1.9.0\_gcc-4.5-sp1

## Obtaining the Files

```
cp /discover/nobackup/jkouatch/pythonTrainingGSFC/Examples/iPythonNotebook  
/iPythonNotebook_SSS0.ipynb .
```

## Launching iPython Notebook

```
ipython notebook --pylab inline
```

## What we are going to cover:

- I. Execute Python syntax
- II. Inline plotting
- III. Run Python scripts
- IV. Access remote file
- V. Run Shell commands
- VI. Do symbolic computations
- VII. Save/print your work

## Basic Python Syntax

```
In [3]: a = 2000  
        b = 1.5  
        c = a*b  
        print c
```

```
3000.0
```

```
In [5]: d = 'NASA'  
e = 'Goddard Space Flight Center'  
f = d+ " " +e  
print f + " , SSS0-Code 610.3"
```

```
NASA Goddard Space Flight Center , SSS0-Code 610.3
```

```
In [9]: import numpy as np  
from time import *  
  
n = 1000  
  
A = np.random.rand(n,n)  
B = np.random.rand(n,n)  
  
beg = time()  
AB = np.dot(A,B)  
end = time()  
  
print 'Time for Matrix Multi.:', 'AB'+str(np.shape(AB)), '=' , 'A'+str(np.shape(A)), 'B'+
```

```
Time for Matrix Multi.: AB(1000, 1000) = A(1000, 1000) B(1000, 1000) is 8.2215278148
```

## Getting help

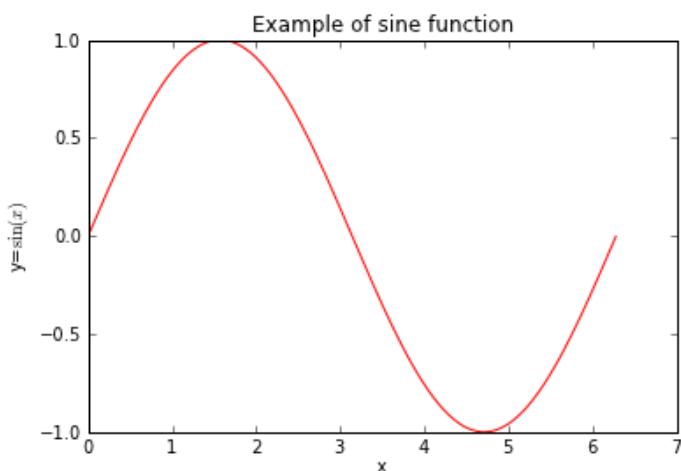
```
In [10]: numpy?
```

```
In [11]:
```

## Plotting

```
In [12]: import math  
x = np.arange(0,2*math.pi,0.01)  
y = np.sin(x)  
plot(x,y,'r',label=r'$\sin(x)$')  
xlabel('x')  
ylabel('y=$\sin(x)$')  
title('Example of sine function')
```

```
Out[12]: <matplotlib.text.Text at 0x2abba0112710>
```



```
In [14]: import numpy as np
import matplotlib.pyplot as plt

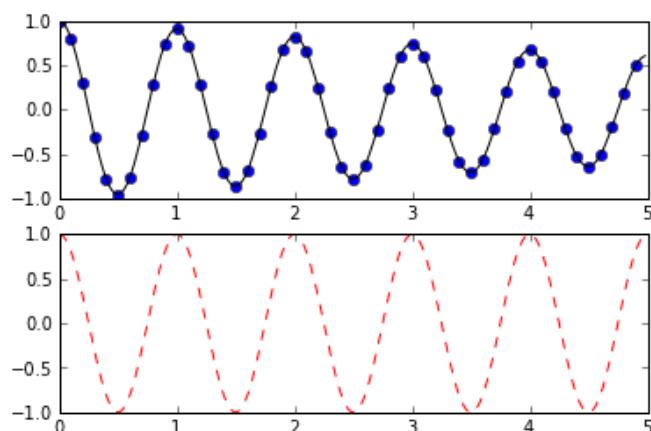
def f(t):
    return np.exp(-0.1*t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')

#plt.savefig('fig_twoFiguresAxes.png')
plt.show()
```



```
In [15]: import matplotlib.pyplot as plt          # pyplot module import
from mpl_toolkits.basemap import Basemap    # basemap import
import numpy as np                          # Numpy import

# Cities names and coordinates
cities = ['London', 'New York', 'Madrid', 'Cairo', 'Moscow', 'Delhi', 'Dakar']
lat = [51.507778, 40.716667, 40.4, 30.058, 55.751667, 28.61, 14.692778]
lon = [-0.128056, -74, -3.683333, 31.229, 37.617778, 77.23, -17.446667]

# orthogonal projection of the Earth
m = Basemap(projection='ortho', lat_0=45, lon_0=10, resolution='l', area_thresh=1000)

#m.bluemarble()

# draw the borders of the map
m.drawmapboundary()
# draw the coasts borders and fill the continents
m.drawcoastlines()
m.fillcontinents(color='coral')

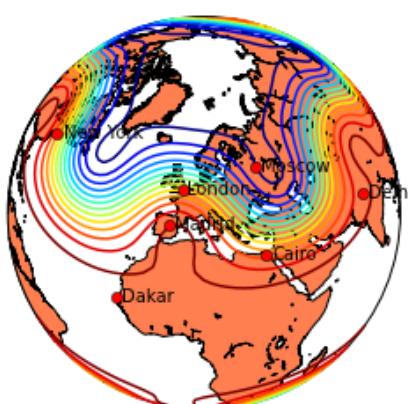
# map city coordinates to map coordinates
x, y = m(lon, lat)

# draw a red dot at cities coordinates
plt.plot(x, y, 'ro')

# for each city,
for city, xc, yc in zip(cities, x, y):
    # draw the city name in a yellow (shaded) box
    plt.text(xc+250000, yc-150000, city)

# make up some data on a regular lat/lon grid.
nlats = 73; nlons = 145; delta = 2.*np.pi/(nlons-1)
lats = (0.5*np.pi-delta*np.indices((nlats,nlons))[0,:,:])
lons = (delta*np.indices((nlats,nlons))[1,:,:])
wave = 0.75*(np.sin(2.*lats)**8*np.cos(4.*lons))
mean = 0.5*np.cos(2.*lats)*((np.sin(2.*lats))**2 + 2.)
# compute native map projection coordinates of lat/lon grid.
x, y = m(lons*180./np.pi, lats*180./np.pi)
# contour data over the map.
CS = m.contour(x,y,wave+mean,15,linewidths=1.5)

#plt.savefig('fig_dataOverMap.png')
plt.show()
```



## Listing iPython Built in commands

In [16]: `lsmagic`

```
Available line magics:  
alias %alias_magic %autocall %automagic %bookmark %cd %clear %colors  
%config %connect_info %debug %dhist %dirs %doctest_mode %ed %edit %env  
%gui %hist %history %install_default_config %install_ext %install_profiles  
%killbgscripts %less %load %load_ext %loadpy %logoff %logon %logstart  
%logstate %logstop %lsmagic %macro %magic %man %more %notebook %page  
%pastebin %pdb %pdef %pdoc %pfile %pinfo %pinfo2 %popd %pprint  
%precision %profile %prun %psearch %psource %pushd %pwd %pycat %pylab  
%qtconsole %quickref %recall %rehashx %reload_ext %rep %rerun %reset  
%reset_selective %run %save %sc %store %sx %system %tb %time %timeit  
%unalias %unload_ext %who %who_ls %whos %xdel %xmode
```

```
Available cell magics:
```

```
%%! %%bash %%capture %%file %%perl %%prun %%ruby %%script %%sh %%sx  
%%system %%timeit
```

Automagic is ON, % prefix IS NOT needed for line magics.

In [ ]: `%hist`

In [20]: %whos

Variable	Type	Data/Info
A	ndarray	1000x1000: 1000000 elems,
type `float64`	, 8000000 bytes (7 Mb)	
AB	ndarray	1000x1000: 1000000 elems,
type `float64`	, 8000000 bytes (7 Mb)	
B	ndarray	1000x1000: 1000000 elems,
type `float64`	, 8000000 bytes (7 Mb)	
Basemap	type	<class
'mpl_toolkits.basemap.Basemap'>		
CS	matplotlib.contour.QuadContourSet	
<matplotlib.contour.QuadC<...>et instance at 0x306dab8>		
a	int	2000
accept2dyear	int	1
altzone	int	14400
asctime	builtin_function_or_method	<built-in function asctime>
b	float	1.5
beg	float	1368121565.36
c	float	3000.0
cities	list	n=7
city	str	Dakar
clock	builtin_function_or_method	<built-in function clock>
ctime	builtin_function_or_method	<built-in function ctime>
d	str	NASA
daylight	int	1
delta	float	0.0436332312999
end	float	1368121573.58
gmtime	builtin_function_or_method	<built-in function gmtime>
lat	list	n=7
lats	ndarray	73x145: 10585 elems, type
`float64`	, 84680 bytes	
localtime	builtin_function_or_method	<built-in function localtime>
lon	list	n=7
lons	ndarray	73x145: 10585 elems, type
`float64`	, 84680 bytes	
m	Basemap	
<mpl_toolkits.basemap.Bas<...>object at 0x2abba18e6990>		
mktimes	builtin_function_or_method	<built-in function mktime>
n	int	1000
nlats	int	73
nlons	int	145
sleep	builtin_function_or_method	<built-in function sleep>
strftime	builtin_function_or_method	<built-in function strftime>
strptime	builtin_function_or_method	<built-in function strptime>
struct_time	type	<type 'time.struct_time'>
t1	ndarray	50: 50 elems, type
`float64`	, 400 bytes	
t2	ndarray	250: 250 elems, type
`float64`	, 2000 bytes	
time	builtin_function_or_method	<built-in function time>
timezone	int	18000
tzname	tuple	n=2
tzset	builtin_function_or_method	<built-in function tzset>
wave	ndarray	73x145: 10585 elems, type
`float64`	, 84680 bytes	
x	ndarray	73x145: 10585 elems, type
`float64`	, 84680 bytes	
xc	float	3530485.10022
y	ndarray	73x145: 10585 elems, type
`float64`	, 84680 bytes	

## Running local Python Scripts

```
In [1]: %load ./H5Py/h5Writing.py
```

```
In [ ]: #!/usr/bin/env python

#-----
# Load modules
#-----
import h5py
import numpy as np
from numpy.random import uniform

# gzip compression flag
comp = 6

#-----
# Creating the file
#-----
hFid = h5py.File('myFile.h5', 'w')

#-----
# File attributes
#-----
hFid.attrs['source']      = 'H5Py Tutorial'
hFid.attrs['history']     = 'Create for GSFC on March 25, 2013'
hFid.attrs['description'] = 'Sample HDF5 file'

#-----
# Defining the dimensions
#-----
lat = np.arange(-90,91,2.0)
dset = hFid.require_dataset('lat', shape=lat.shape, dtype=np.float32, compression=comp)
dset[...] = lat
dset.attrs['name'] = 'latitude'
dset.attrs['units'] = 'degrees north'

lon = np.arange(-180,180,2.5)
dset = hFid.require_dataset('lon', shape=lon.shape, dtype=np.float32, compression=comp)
dset[...] = lon
dset.attrs['name'] = 'longitude'
dset.attrs['units'] = 'degrees east'

lev = np.arange(0,72,1)
dset = hFid.require_dataset('lev', shape=lev.shape, dtype=np.int, compression=comp)
dset[...] = lev
dset.attrs['name'] = 'vertical levels'
dset.attrs['units'] = 'hPa'

time = np.arange(0,5,1)
dset = hFid.require_dataset('time', shape=time.shape, maxshape=(None), dtype=np.float32, compression=comp)
dset[...] = time
dset.attrs['name'] = 'time'
dset.attrs['units'] = 'hours since 2013-01-01 00:00:00.0'
dset.attrs['calendar'] = 'gregorian'

#-----
# Creating variables and Setting attributes
#-----
arr = np.zeros((5,lev.size,lat.size,lon.size))
arr[0:5,:,:,:] = 300*uniform(size=(5,lev.size,lat.size,lon.size))
dset = hFid.require_dataset('temp', shape=arr.shape, dtype=np.float32, compression=comp)
dset[...] = arr
dset.attrs['name'] = 'temperature'
dset.attrs['units'] = 'K'
```

```
In [54]: %load ./H5Py/h5Reading.py
```

```
In [56]: #!/usr/bin/env python

#-----
# Load modules
#-----
import h5py
import numpy as np

# gzip compression flag
comp = 6

#-----
# Creating the file
#-----
hFid = h5py.File('myFile.h5', 'r')
print hFid.name
print hFid.keys()

lev  = hFid['lev'].value
lat  = hFid['lat'].value
lon  = hFid['lon'].value
time = hFid['time'].value

temp1 = hFid['temp'].value
print temp1[0,0,0,0], temp1[4,6,7,10]

temp2 = hFid['3D_Data']['temp'].value
print temp2[0,0,0,0], temp2[4,6,7,7]

#-----
# Closing the file
#-----
hFid.close()
```

```
/ ['2D_Data', '3D_Data', 'lat', 'lev', 'lon', 'temp', 'time']
101.414 139.848
101.41419495 176.190046124
```

```
In [ ]: %run table_demo.py
```

## Running Shell Commands

```
In [ ]: !ls
```

```
In [ ]: !ping www.nasa.gov
```

## Accessing remote files

```
In [27]: %load http://matplotlib.org/mpl_examples/pylab_examples/fill_demo.py
```

```
In [ ]: #!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0.0, 1.01, 0.01)
s = np.sin(2*2*np.pi*t)

plt.fill(t, s*np.exp(-5*t), 'r')
plt.grid(True)
plt.show()
```

```
In [28]: %load http://matplotlib.org/mpl_examples/api/radar_chart.py
```

In [29]:

```
"""
Example of creating a radar chart (a.k.a. a spider or star chart) [1]_.

Although this example allows a frame of either 'circle' or 'polygon', polygon
frames don't have proper gridlines (the lines are circles instead of polygons).
It's possible to get a polygon grid by setting GRIDLINE_INTERPOLATION_STEPS in
matplotlib.axis to the desired number of vertices, but the orientation of the
polygon is not aligned with the radial axes.

.. [1] http://en.wikipedia.org/wiki/Radar_chart
"""

import numpy as np

import matplotlib.pyplot as plt
from matplotlib.path import Path
from matplotlib.spines import Spine
from matplotlib.projections.polar import PolarAxes
from matplotlib.projections import register_projection


def radar_factory(num_vars, frame='circle'):
    """Create a radar chart with `num_vars` axes.

    This function creates a RadarAxes projection and registers it.

    Parameters
    -----
    num_vars : int
        Number of variables for radar chart.
    frame : {'circle' | 'polygon'}
        Shape of frame surrounding axes.

    """
    # calculate evenly-spaced axis angles
    theta = 2*np.pi * np.linspace(0, 1-1./num_vars, num_vars)
    # rotate theta such that the first axis is at the top
    theta += np.pi/2

    def draw_poly_patch(self):
        verts = unit_poly_verts(theta)
        return plt.Polygon(verts, closed=True, edgecolor='k')

    def draw_circle_patch(self):
        # unit circle centered on (0.5, 0.5)
        return plt.Circle((0.5, 0.5), 0.5)

    patch_dict = {'polygon': draw_poly_patch, 'circle': draw_circle_patch}
    if frame not in patch_dict:
        raise ValueError('unknown value for `frame`: %s' % frame)

    class RadarAxes(PolarAxes):

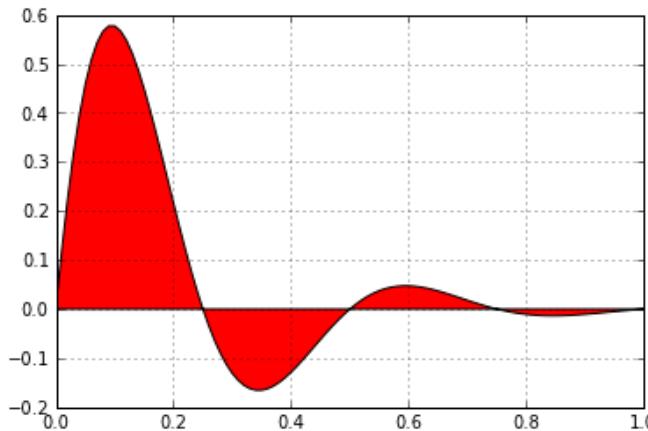
        name = 'radar'
        # use 1 line segment to connect specified points
        RESOLUTION = 1
        # define draw_frame method
        draw_patch = patch_dict[frame]

        def fill(self, *args, **kwargs):
            """Override fill so that line is closed by default"""
            closed = kwargs.pop('closed', True)
            # ... (rest of the fill implementation)
```

```
In [25]: #!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0.0, 1.01, 0.01)
s = np.sin(2*2*np.pi*t)

plt.fill(t, s*np.exp(-5*t), 'r')
plt.grid(True)
plt.show()
```



```
In [ ]: from IPython.display import YouTubeVideo
YouTubeVideo('iwVvqwLDsJo')
```

## LaTex

$$x = \sum_{i=1}^n (a_i - b_i)^2$$

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k x} dx$$

```
In [31]: from IPython.display import display, Math, Latex
display(Math(r'F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k x} dx'))
```

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k x} dx$$

## Symbolic Computations

```
In [36]: %load_ext sympyprinting
import sympy as sym
from sympy import *
x, y, z = sym.symbols("x y z")
```

```
/usr/local/other/SLES11/SIV0-PyD/1.9.0/lib/python2.7/site-packages/IPython
/extensions/sympyprinting.py:119: UserWarning: The sympyprinting extension in
IPython is deprecated, use sympy.interactive.ipythonprinting
warnings.warn("The sympyprinting extension in IPython is deprecated, "
```

```
In [37]: eq1 = Rational(3,2)*pi + exp(I*x) / (x**2 + y)
eq1
```

```
Out[37]:  $\frac{3}{2}\pi + \frac{e^{ix}}{x^2 + y}$ 
```

```
In [38]: eq2 = ((x+y)**2 * (x+1))
eq2
```

```
Out[38]:  $(x + 1)(x + y)^2$ 
```

```
In [39]: diff(cos(x**2)**2 / (1+x), x)
```

```
Out[39]:  $-4 \frac{x \sin(x^2) \cos(x^2)}{x + 1} - \frac{\cos^2(x^2)}{(x + 1)^2}$ 
```

```
In [40]: (1/cos(x)).series(x, 0, 10)
```

```
Out[40]:  $1 + \frac{1}{2}x^2 + \frac{5}{24}x^4 + \frac{61}{720}x^6 + \frac{277}{8064}x^8 + O(x^{10})$ 
```

```
In [41]: limit(sin(x)/x, x, 0)
```

```
Out[41]: 1
```

```
In [42]: integrate(log(y), y)
```

```
Out[42]:  $y \log(y) - y$ 
```

```
In [43]: integrate(exp(-x**2), (x, -oo, oo))
```

```
Out[43]:  $\sqrt{\pi}$ 
```

```
In [44]: simplify('cos(x)**2+sin(x)**2')
```

```
Out[44]: 1
```

```
In [ ]:
```